Syntactic parsing.[1]

Victor Kitov

v.v.kitov@yandex.ru

[1] With materials used from "Speech and Language Processing", D. Jurafsky and J. H. Martin.

Constituents

- Morphology - words
- Syntax - in Greek *setting out together or arrangement*.
- Semantics - meaning

Constituents

- Constituents: groups of words behaving as a single units
    - Examples of NP constituents:
        - Harry the Horse
        - a high-class spot
        - such as Mindy's
        - the Broadway coppers
        - the reason he comes into the Hot Box
        - they
        - three parties from Brooklyn

Constituents

- Constituents appear in similar syntactic contexts
  - **three parties from Brooklyn** arrive. . .
  - a high-class spot **such as Mindy's** attracts. . .
  - **the Broadway coppers** love. . .
  - **they** sit
- Constituents can be moved inside the sentence:
  - **On September seventeenth**, I'd like to fly from Atlanta to Denver
  - I'd like to fly **on September seventeenth** from Atlanta to Denver
  - I'd like to fly from Atlanta to Denver **on September seventeenth**

Examples of grammar

$$\begin{aligned} NP &\rightarrow Det\ Nominal \\ NP &\rightarrow ProperNoun \\ Nominal &\rightarrow Noun \mid Nominal\ Noun \end{aligned}$$

$$\begin{aligned} Det &\rightarrow a \\ Det &\rightarrow the \\ Noun &\rightarrow flight \end{aligned}$$

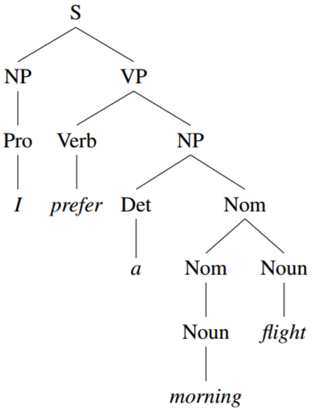
- Rules, generating language, may be nested.
- Symbols:
    - terminal: express final words
        - a, the, flight, etc.
    - non-terminal (express language abstractions)
        - NP, VP, PP, etc.
- Non-terminals, generating terminals, correspond to parts of speech.

More example rules

- S $\rightarrow$ NP VP (I prefer a morning flight)
- VP $\rightarrow$ Verb NP (prefer a morning flight)
- VP $\rightarrow$ Verb NP PP (leave Boston in the morning)
- VP $\rightarrow$ Verb PP (leaving on Thursday)
- PP $\rightarrow$ Preposition NP (from Los Angeles)

Parse trees

- Sample parse tree:



- Bracketed notation of parse trees (common in datasets)
  - [S [NP [Pro **I**]] [VP [V **prefer**] [NP [Det **a**] [Nom [N **morning**] [Nom [N **flight**]]]]]]

Definitions

N	a set of **non-terminal symbols** (or **variables**)
Σ	a set of **terminal symbols** (disjoint from N)
R	a set of **rules** or productions, each of the form $A \rightarrow \beta$,
	where A is a non-terminal,
	β is a string of symbols from the infinite set of strings $(\Sigma \cup N)*$
S	a designated **start symbol** and a member of N

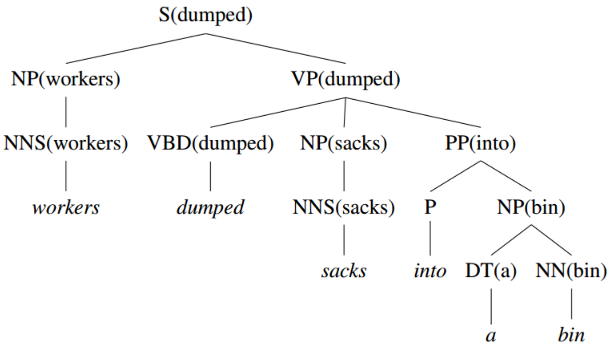
Notation:

- Capital letters like A, B, and S Non-terminals
- S The start symbol
- ε - empty symbol
- Lower-case Greek letters like α, β, and γ strings drawn from $(\Sigma \cup N)^*$
- Lower-case Roman letters like u, v, and w strings of terminals

Applications of grammar

- Language generation
  - A language, generated by grammar G is a subset of stings from Σ^* that can be derived using rules of G.
- Parsing of existing sentences
  - Syntactic parsing - mapping from a string of words to its parse tree
- Treebank - syntactically annotated corpus.
  - Penn Treebank project has treebanks from the Brown, Switchboard, ATIS, and Wall Street Journal corpora of English
  - We can extract grammar from treebank
    - it will be flat (many long rules), so post-processing is needed

Syntax parsing with head words



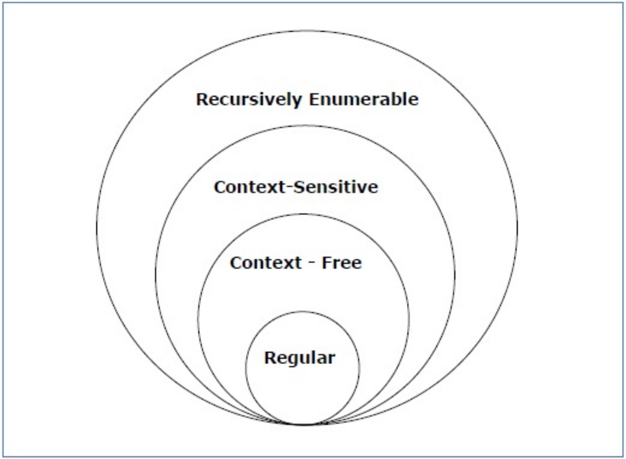
- Can incorporate head word into grammar
- More commonly: make regular parse and then assign head words with simple rules.

Grammars equivalence

- Grammars are weakly equivalent, if they generate the same set of strings.
- Grammars are strongly equivalent, if they generate the same set of strings and every sentence has the same parse tree
  - up to renaming non-terminals

Chomsky grammar classification

Chomsky grammar classification



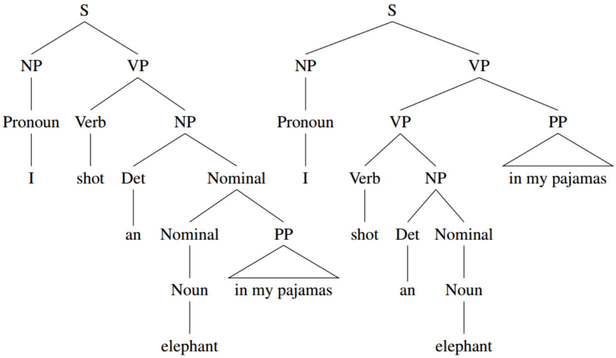
Chomsky grammar classification

- Regular language:
    - allowed rules: $X \to a$ or $X \to aY$, $X, Y \in N$, $a \in T$
    - $S \to \varepsilon$ is allowed only if S doesn't appear on the right side of any rule
    - recognizable by finite state automation or regular expressions.
- Context-free grammars:
    - allowed rules: $X \to \gamma$, $X \in N$, $\gamma \in (T \cup N)^*$
- Context-sensitive grammars:
    - $\alpha B\beta \to \alpha\gamma\beta$, where $B \in N$, $\alpha, \gamma, \beta \in (T \cup N)^*$
    - α, β may be empty, γ must be non-empty
    - $S \to \varepsilon$ is allowed only if S doesn't appear on the right side of any rule
- recursively enumerable grammar - no restrictions on rules
    - generate the languages that are recognized by a Turing machine.

Ambiguity

Examples of structural ambiguity:

- We saw [the Eiffel Tower flying to Paris].
- We saw [the Eiffel Tower] flying to Paris.
- [old [men and women]] <->[old men] and [women]



Chomsky normal form

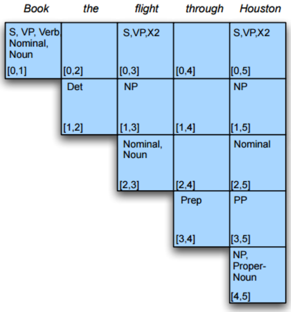
- Every context free grammar can be produced in Chomsky normal form (CNF)
- CYK (Cocke-Kasami-Younger) parser deals only with CNF context free grammars.
- In CNF all rules can be only:
    - A->v (non-teminal to terminal)
    - A->BC (non-terminal to 2 non-terminals)
- Example of CNF conversion:
    - A $\rightarrow$ B C D may be convereted to
    - A $\rightarrow$ B X
    - X $\rightarrow$ C D

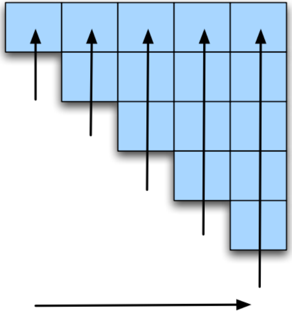
Conversion to CNF

- Mix of terminals and non-terminals on right hand side:
  - INF-VP $\to$ to VP replace by
    - INF-VP $\to$ TO VP
    - TO $\to$ to
- Single non-terminal on right hand side $A \to B$:
  - if A can be converted to B by some sequence of rules
  - for every rule $A \to \gamma$ add $B \to \gamma$
- More than 2 non-terminals on right hand side $A \to BC\gamma$
  - replace with:
    - $A \to X\gamma$
    - $X \to BC$

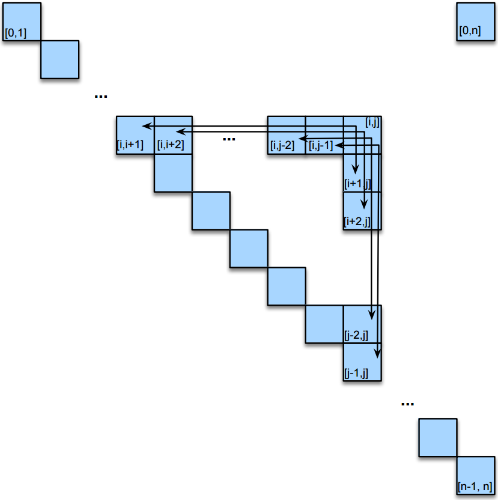
CYK matrix

- Positions of the sentence:
    - 0 Book 1 that 2 flight 3.
- Work with $(N+1)x(N+1)$ matrix M
    - upper-triangular position are needed
    - $M[i,j]$ stands for parsing of sentence[i:j]
    - $M[i,j]$ if formed from $M[i,k]$ and $M[k,j]$ recurrently





CYK illustration



CYK algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

  for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
    for all $\{A \mid A \rightarrow words[j] \in grammar\}$
      $table[j-1, j] \leftarrow table[j-1, j] \cup A$
    for $i \leftarrow$ **from** $j-2$ **downto** 0 **do**
      for $k \leftarrow i+1$ **to** $j-1$ **do**
        for all $\{A \mid A \rightarrow BC \in grammar$ **and** $B \in table[i,k]$ **and** $C \in table[k,j]\}$
          $table[i,j] \leftarrow table[i,j] \cup A$

- This is a recognizer
- To recognize sentence it needs to find S in matrix[0,N]
- To make it a parser we need to add backtracking
  - add from which elements each new element can be derived

Restoring original grammar

- We can augment CYK parser to deal with rules $A \to B$
- Rules $A \to BC$, $A \to wC$, $A \to Cw$ can be restored by merging CNF rules

Partial parsing

- Partial (shallow) parsing - parse only lower layers of tree
  - chunk sentence into segments
  - [NP The morning flight] [PP from] [NP Denver] [VP has arrived.]
  - some words may not be covered
  - e.g. when only NP are interesting: [NP The morning flight] from [NP Denver] has arrived.
- Applications: information extraction from most informative sentence parts.
- Chunking is performed with sequence labelling (aka POS tagging)
- IOB notation:
  - B-beginning of next chunk
  - I-inside current chunk
  - O-outside any chunk

Examples of partial parsing

- [NP The morning flight] [PP from] [NP Denver] [VP has arrived.]

B	I	I	B	B	B	I
The	morning	flight	from	Denver	has	arrived

- [NP The morning flight] from [NP Denver] has arrived.

B	I	I	O	B	O	O
The	morning	flight	from	Denver	has	arrived

- We can automatically deduce the endings of each chunk

Examples of partial parsing with chunk type

- [NP The morning flight] [PP from] [NP Denver] [VP has arrived.]

B_NP	I_NP	I_NP	B_PP	B_NP	B_VP	I_VP
The	morning	flight	from	Denver	has	arrived

- [NP The morning flight] from [NP Denver] has arrived.

B_NP	I_NP	I_NP	O	B_NP	O	O
The	morning	flight	from	Denver	has	arrived

- We can automatically deduce the endings of each chunk
- Total number of tags $2n+1$, where n is the number of chunk types.

Training

- Use treebanks to generate training set
- Use sequence labelling algorithms
- Features:
    - look on words at positions -2,-1,0,1,2
    - get
        - words themselves
        - parts of speech
        - previous classifications at positions -2,-1.

Evaluation

$$\text{Precision} = \frac{\text{Number of correct chunks given by system}}{\text{Total number of chunks given by system}}$$

$$\text{Recall} = \frac{\text{Number of correct chunks given by system}}{\text{Total number of actual chunks in the text}}$$

Overall measure (weighted harmonic mean)

$$F_\beta = \frac{1}{\frac{\beta^2}{\beta^2+1}\frac{1}{R} + \frac{1}{\beta^2+1}\frac{1}{P}}$$

or simply (uniform harmonic mean)

$$F_1 = \frac{1}{\frac{1}{2}\frac{1}{R} + \frac{1}{2}\frac{1}{P}}$$

State-of-the-art results

- State-of-the-art results:
    - NP chunking, English language: 0.96
    - NP,VP,PP,ADVP,SBAR,ADJP chunking, English language: 0.92 - 0.94
- Problems:
    - POS tagging accuracy
    - inconsistencies of correct labels generation from parse trees
    - ambiguities:
        - [NP Late arrivals and departures] are commonplace during winter.
        - [NP Late arrivals] and [NP cancellations] are commonplace during winter.
        - semantic information is needed to make correct chunking!

PCFG

Probabilistic context free grammar:

N	set of non-terminal symbols
Σ	set of terminal symbols
R	set of rules $A \to \beta\,[p]$, $A \in N$, $\beta \in (\Sigma \cup N)*$
S	start symbol

- We can estimate probabilities from treebanks.
- Independence assumption: rules application probabilities don't depend on already applied rules.

Probabilistic parsing

- S-observed sentence, T-unobserved parse tree
- We are interested to find most probable parse:

$$T = \arg\max_{T} p(T|S)$$

- Probability of sentence:

$$P(S) = \sum_{T} p(S|T)$$

Probabilistic CYK algorithm

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                          and its probability
  for j ← from 1 to LENGTH(words) do
     for all  A; A → words[j] ∈ grammar
        table[j−1, j, A] ← P(A → words[j])
     for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
              for all  A; A → BC ∈ grammar,
                         and table[i,k,B] > 0 and table[k,j,C] > 0
                 if (table[i,j,A] < P(A → BC)  table[i,k,B]  table[k,j,C]) then
                    table[i,j,A] ← P(A → BC)  table[i,k,B]  table[k,j,C]
                    back[i,j,A] ←  k,B,C
  return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```